Exhibit 13 to Complaint
Intellectual Ventures I LLC and Intellectual Ventures II LLC

Example American Count 6 Systems and Services
U.S. Patent No. 7,257,582 ("'582 Patent")

The Accused Systems and Services include without limitation American systems and services that utilize Hadoop; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future American systems and services that have the same or substantially similar features as the specifically identified systems and services ("Example American Count 6 Systems and Services" or "American Systems and Services").[1]

On information and belief, the American Systems and Services use Hadoop in its private cloud(s). For example, the following job postings show that Hadoop is used by American Airlines.

- Example of a Senior Data Engineer at American Airlines whose position mentions use of Hadoop to extract data and analyze programs. https://www.linkedin.com/in/sriram1993/ (Last accessed on 9/18/24)
- Example of a Data Engineer at American Airlines whose position mentions Hadoop. https://www.linkedin.com/in/madhu-sudhan-reddy-y-b3897b129/ (Last accessed on 9/18/24)
- Example of a Big Data Engineer at American Airlines whose position mentions Hadoop. https://www.linkedin.com/in/krishna-karthika-katragadda-149786306/ (Last accessed on 9/18/24)
- Example of a Data Engineer at American Airlines whose position mentions Hadoop. https://www.linkedin.com/in/ramya-arikapudi-5290a0186/ ("[r]esponsible for building scalable distributed data solutions using Hadoop").

As another example, American has announced cloud migration of legacy technology and efforts to modernize its mainframes and servers. Source: https://dxc.com/sg/en/insights/customer-stories/american-airlines-cloud-data-automation.

On information and belief, other evidence confirms American uses Hadoop technology.

[1] For the avoidance of doubt, Plaintiffs do not accuse public clouds of American if those services are provided by a cloud provider with a license to Plaintiffs' patents that covers American's activities. IV will provide relevant license agreements for cloud providers in discovery. To the extent any of these licenses are relevant to American's activities, Plaintiffs will meet and confer with American about the impact of such license(s).

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and stor age. Read less

American Airlines

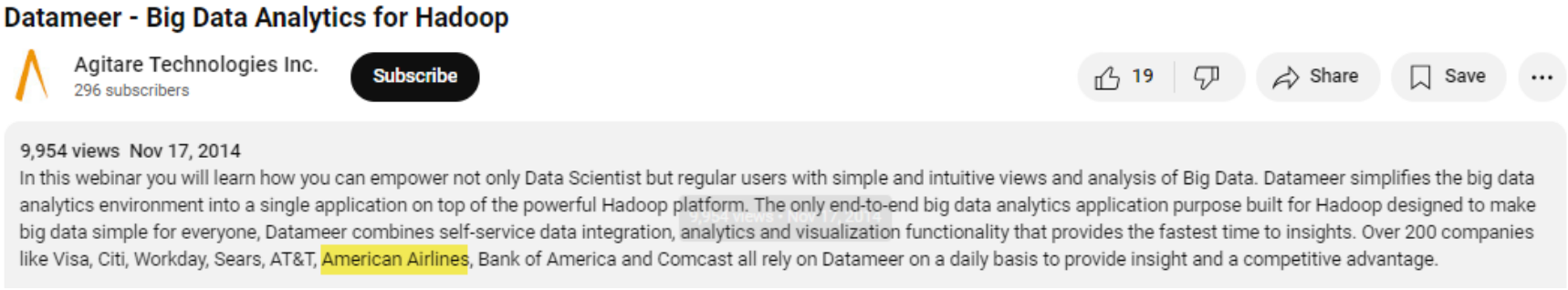
Technologies used by the company: 1,293

Source: https://www.zoominfo.com/tech/24689/apache-hadoop-tech-from-transportation-airline-industry-in-us-by-revenue.

Recognizing the tremendous value of Hadoop for Big Data processing and analytics, many companies invest time and resources in this technology, seeing this as a necessary step towards the sustainable development of their businesses. Some of the notable users of Hadoop include Amazon.com, American Airlines, Apple, eBay, Facebook, Hewlett-Packard, IBM, LinkedIn, Microsoft, Twitter, Yahoo!.

Source: https://www.elinext.com/blog/why-hadoop-for-big-data/.

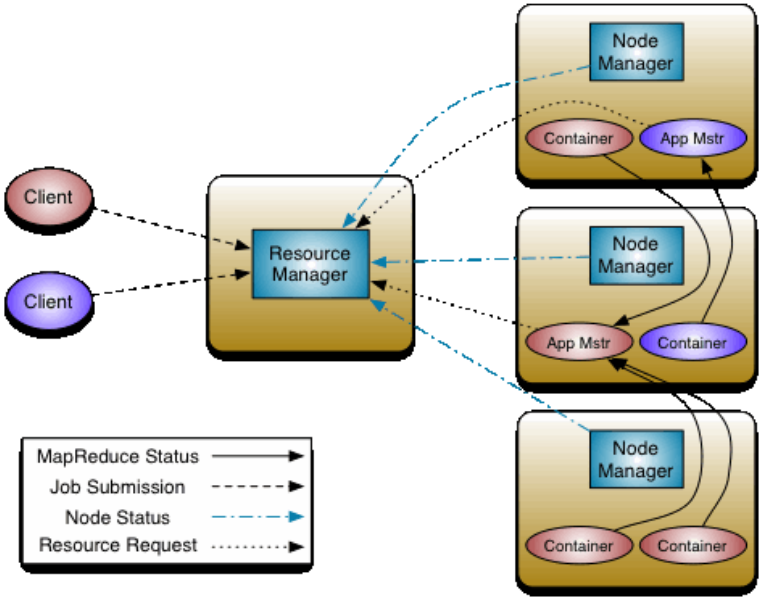
Source: https://www.slideshare.net/slideshow/big-data-and-hadoop-42639482/42639482#39.



Source: https://www.youtube.com/watch?v=X26ghqOnok0.

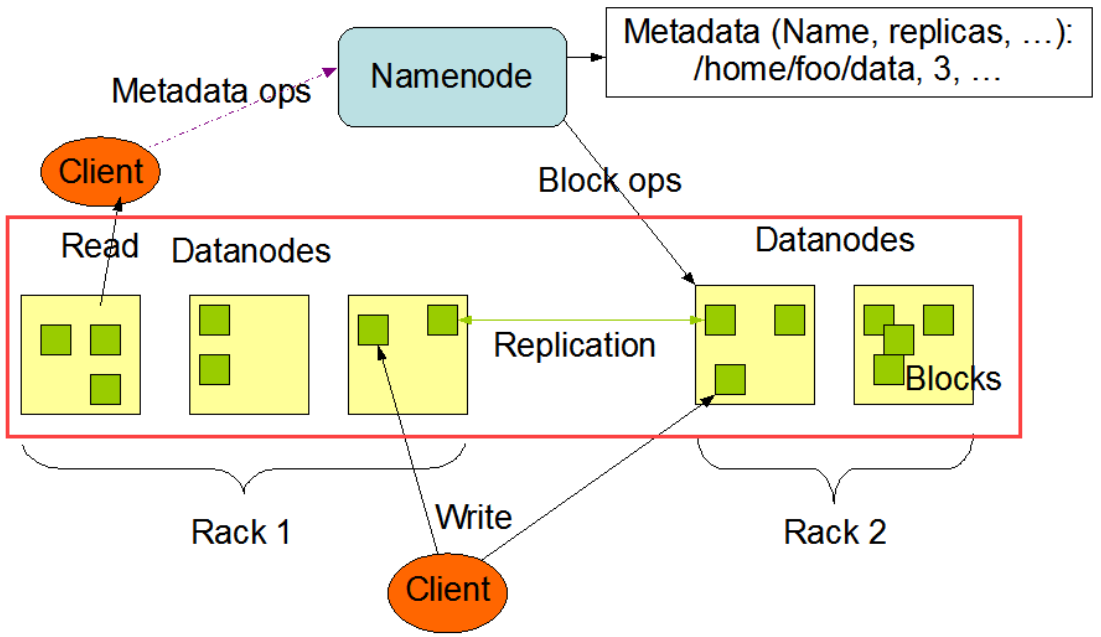
U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
1. A method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks, the method comprising the steps of:	To the extent this preamble is limiting, on information and belief, the American Count 6 Systems and Services practice a method of effecting on a preexisting input file a computer-executable process comprised of a plurality of subtasks. On information and belief, Hadoop includes MapReduce, a software framework that processes multiple map tasks in parallel. An input data set is split into multiple chunks and each chunk is processed by a map task. **Overview** Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.[2]

[2] All citations in this claim chart were publicly accessible as of the filing of the Complaint.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
	 Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
(a) automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions;	On information and belief, the American Count 6 Systems and Services practice automatically determining file allocation and logically subdividing records of said input file into a plurality of partitions. On information and belief, Hadoop's InputFormat describes the input-specification for a MapReduce job. The input file is logically split into multiple InputSplit instances.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
	Job Input InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the InputFormat of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical InputSplit instances, each of which is then assigned to an individual Mapper. 3. Provide the RecordReader implementation used to glean input records from the logical InputSplit for processing by the Mapper. The default behavior of file-based InputFormat implementations, typically sub-classes of FileInputFormat, is to split the input into *logical* InputSplit instances based on the total size, in bytes, of the input files. However, the FileSystem blocksize of the input files is treated as an upper bound for input splits. A lower bound on the split size can be set via mapreduce.input.fileinputformat.split.minsize. Clearly, logical splits based on input-size is insufficient for many applications since record boundaries must be respected. In such cases, the application should implement a RecordReader, who is responsible for respecting record-boundaries and presents a record-oriented view of the logical InputSplit to the individual task. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
(b) distributing descriptions of all of said partitions to each of a plurality of subtask processors	On information and belief, the American Count 6 Systems and Services practice distributing descriptions of all of said partitions to each of a plurality of subtask processors. On information and belief, Hadoop's InputSplit represents data that presents a byte oriented view of the input. A recordreader converts the byte oriented view of the data to a record oriented view and presents it to a mapper. Mappers run inside a compute node. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
	Job Input InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the `InputFormat` of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical `InputSplit` instances, each of which is then assigned to an individual `Mapper`. 3. Provide the `RecordReader` implementation used to glean input records from the logical `InputSplit` for processing by the `Mapper`. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. If `TextInputFormat` is the `InputFormat` for a given job, the framework detects input-files with the *.gz* extensions and automatically decompresses them using the appropriate `CompressionCodec`. However, it must be noted that compressed files with the above extensions cannot be *split* and each compressed file is processed in its entirety by a single mapper. **InputSplit** InputSplit represents the data to be processed by an individual `Mapper`. Typically `InputSplit` presents a byte-oriented view of the input, and it is the responsibility of `RecordReader` to process and present a record-oriented view. FileSplit is the default `InputSplit`. It sets `mapreduce.map.input.file` to the path of the input file for the logical split. **RecordReader** RecordReader reads `<key, value>` pairs from an `InputSplit`. Typically the `RecordReader` converts the byte-oriented view of the input, provided by the `InputSplit`, and presents a record-oriented to the `Mapper` implementations for processing. `RecordReader` thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
	HDFS Architecture  Source: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.
(c) simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective	On information and belief, the American Count 6 Systems and Services practice simultaneously executing at least a respective one of the subtasks of the computer-executable process in each of at least some of said processors on a respective one of the partitions with each subtask reading and processing the respective partition so as to process the respective partition and produce respective subtask output. On information and belief, Multiple map tasks are executed in parallel. Each Maptask processes a respective InputSplit.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
partition so as to process the respective partition and produce respective subtask output and;	**Overview** Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. **Job Input** InputFormat describes the input-specification for a MapReduce job. The MapReduce framework relies on the `InputFormat` of the job to: 1. Validate the input-specification of the job. 2. Split-up the input file(s) into logical `InputSplit` instances, each of which is then assigned to an individual `Mapper`. 3. Provide the `RecordReader` implementation used to glean input records from the logical `InputSplit` for processing by the `Mapper`. Source: Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
	Mapper Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. **How Many Maps?** The number of maps is usually driven by the total size of the inputs, that is, the total number of blocks of the input files. The right level of parallelism for maps seems to be around 10-100 maps per-node, although it has been set up to 300 maps for very cpu-light map tasks. Task setup takes a while, so it is best if the maps take at least a minute to execute. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. On information and belief, Each MapTask generates an intermediate map output. **Overview** Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce *job* usually splits the input data-set into independent chunks which are processed by the *map tasks* in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. **InputSplit** InputSplit represents the data to be processed by an individual Mapper. Typically InputSplit presents a byte-oriented view of the input, and it is the responsibility of RecordReader to process and present a record-oriented view. FileSplit is the default InputSplit. It sets mapreduce.map.input.file to the path of the input file for the logical split. **RecordReader** RecordReader reads <key, value> pairs from an InputSplit. Typically the RecordReader converts the byte-oriented view of the input, provided by the InputSplit, and presents a record-oriented to the Mapper implementations for processing. RecordReader thus assumes the responsibility of processing record boundaries and presents the tasks with keys and values. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. **Mapper** Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
	Data Compression Hadoop MapReduce provides facilities for the application-writer to specify compression for both intermediate map-outputs and the job-outputs i.e. output of the reduces. It also comes bundled with CompressionCodec implementation for the zlib compression algorithm. The gzip, bzip2, snappy, and lz4 file format are also supported. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
(d) thereafter repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis; and	On information and belief, the American Count 6 Systems and Services practice repeating step (c) in at least some of the subtask processors each with another unprocessed partition on a first-come/first-served basis. On information and belief, Hadoop includes a FIFO scheduler. Each mapper processes an InputSplit at a time. For example, a mapper processes one line of the word count application at a time. **Overview** **Overview** The Yarn scheduler is a fertile area of interest with different implementations, e.g., Fifo, Capacity and Fair schedulers. Meanwhile, several optimizations are also made to improve scheduler performance for different scenarios and workload. Each scheduler algorithm has its own set of features, and drives scheduling decisions by many factors, such as fairness, capacity guarantee, resource availability, etc. It is very important to evaluate a scheduler algorithm very well before we deploy in a production cluster. Unfortunately, currently it is non-trivial to evaluate a scheduler algorithm. Evaluating in a real cluster is always time and cost consuming, and it is also very hard to find a large-enough cluster. Hence, a simulator which can predict how well a scheduler algorithm for some specific workload would be quite useful. Source:. **Example: WordCount v1.0** Before we jump into the details, lets walk through an example MapReduce application to get a flavour for how they work. WordCount is a simple application that counts the number of occurrences of each word in a given input set.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**

	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
	
	Walk-through
	
	The `WordCount` application is quite straight-forward.
	
	``
	public void map(Object key, Text value, Context context
	                ) throws IOException, InterruptedException {
	  StringTokenizer itr = new StringTokenizer(value.toString());
	  while (itr.hasMoreTokens()) {
	    word.set(itr.nextToken());
	    context.write(word, one);
	  }
	}
	``
	
	The `Mapper` implementation, via the `map` method, processes one line at a time, as provided by the specified `TextInputFormat`. It then splits the line into tokens separated by whitespaces, via the `StringTokenizer`, and emits a key-value pair of `< <word>, 1>`.
	
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**

	For the given sample input the first map emits:
	`< Hello, 1>`
	`< World, 1>`
	`< Bye, 1>`
	`< World, 1>`
	The second map emits:
	`< Hello, 1>`
	`< Hadoop, 1>`
	`< Goodbye, 1>`
	`< Hadoop, 1>`
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**

	`WordCount` also specifies a `combiner`. Hence, the output of each map is passed through the local combiner (which is same as the `Reducer` as per the job configuration) for local aggregation, after being sorted on the *keys*.
	The output of the first map:
	`< Bye, 1>`
	`< Hello, 1>`
	`< World, 2>`
	The output of the second map:
	`< Goodbye, 1>`
	`< Hadoop, 2>`
	`< Hello, 1>`
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

(e) generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks.	On information and belief, the American Count 6 Systems and Services practice generating at least one output combining all of the subtask outputs and reflecting the processing of all of said subtasks. On information and belief, intermediate values are grouped by the Hadoop framework and passed to a Reducer to determine the final output.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**
	Mapper Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The Hadoop MapReduce framework spawns one map task for each `InputSplit` generated by the `InputFormat` for the job. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to the `Reducer(s)` to determine the final output. Users can control the grouping by specifying a `Comparator` via Job.setGroupingComparatorClass(Class). Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. The `Mapper` outputs are sorted and then partitioned per `Reducer`. The total number of partitions is the same as the number of reduce tasks for the job. Users can control which keys (and hence records) go to which `Reducer` by implementing a custom `Partitioner`. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. **Data Compression** Hadoop MapReduce provides facilities for the application-writer to specify compression for both intermediate map-outputs and the job-outputs i.e. output of the reduces. It also comes bundled with CompressionCodec implementation for the zlib compression algorithm. The gzip, bzip2, snappy, and lz4 file format are also supported. Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

	On information and belief, the Reducer reduces a set of intermediate values to a smaller output.
	
	Reducer
	
	Reducer reduces a set of intermediate values which share a key to a smaller set of values.
	
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
	
	The Reducer implementation, via the reduce method just sums up the values, which are the occurrence counts for each key (i.e. words in this example).
	
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.
	
	WordCount also specifies a combiner. Hence, the output of each map is passed through the local combiner (which is same as the Reducer as per the job configuration) for local aggregation, after being sorted on the *keys*.
	
	The output of the first map:
	
	`< Bye, 1>`
	`< Hello, 1>`
	`< World, 2>`
	
	The output of the second map:
	
	`< Goodbye, 1>`
	`< Hadoop, 2>`
	`< Hello, 1>`
	
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

U.S. Patent No. 7,257,582 (Claim 1)	
Claim(s)	**Example American Count 6 Systems and Services**

	Thus the output of the job is:
	
	`< Bye, 1>`
	`< Goodbye, 1>`
	`< Hadoop, 2>`
	`< Hello, 2>`
	`< World, 2>`
	
	Source: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.